

Projeto – Criando Aplicação Orientada a Objeto + Swing

Neste laboratório irá desenvolver para aplicação Banco que está sendo construída ao longo deste laboratório.

Sugerimos que estes exemplos sejam feitos com uso da IDE Eclipse.

Descrição do Problema

O Sistema proposto é um sistema que terá um CRUD de clientes, ou seja, inclusão, alteração, exclusão e listagem de clientes. Cada cliente terá possibilidade de realizar operações bancárias, tais como:

- Saque
- Depósito
- Transferência entre contas

O Sistema é uma sequência das atividades realizadas até o momento, utilize da boa prática de reutilização de código.

O Sistema deverá ser modularizado conforme a figura 2 deste documento.

Ao final deste documento vai ser anexada a solução para o problema, porém é de grande importância que você tente desenvolver a solução, esta que será usada durante todo o curso.

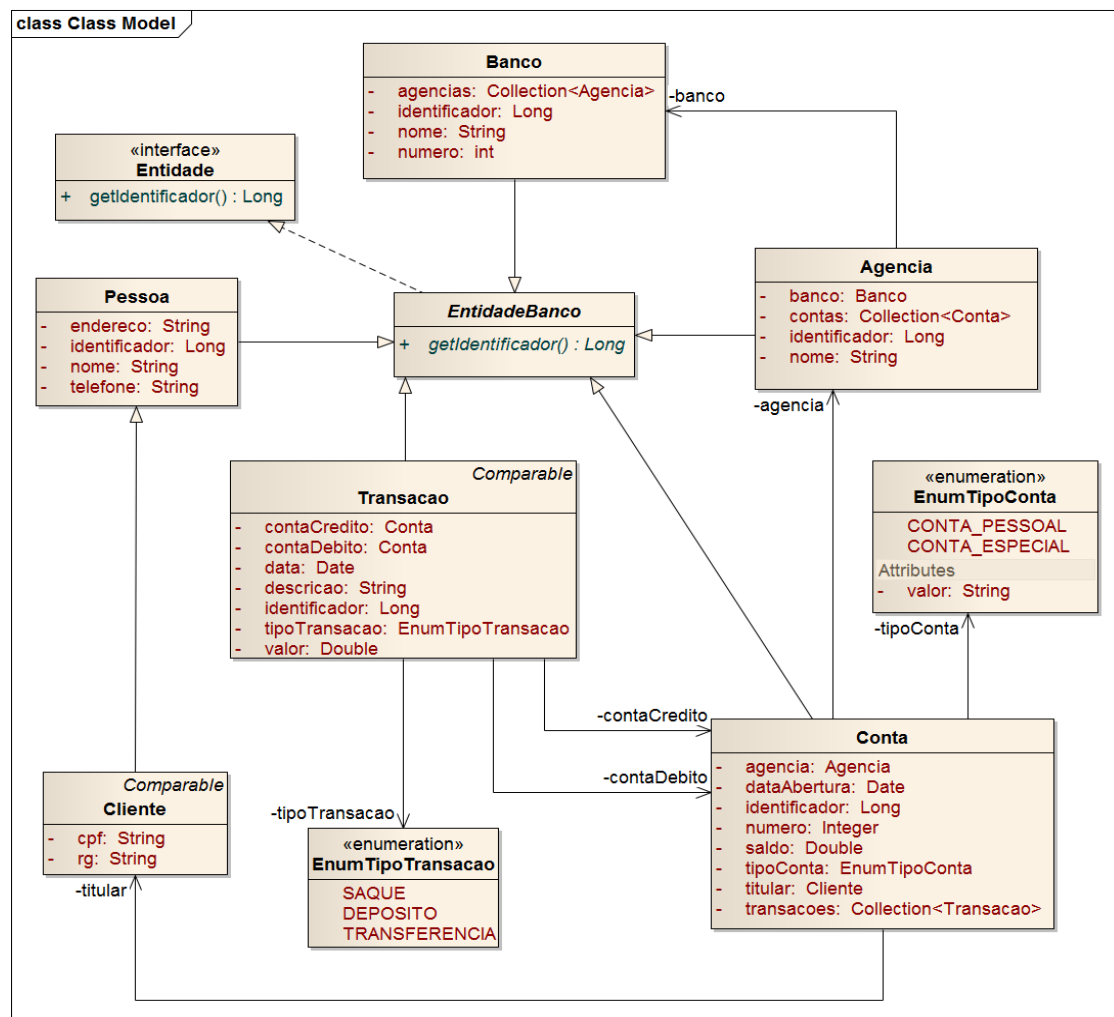


Figura 1 - Diagrama UML para aplicação Banco

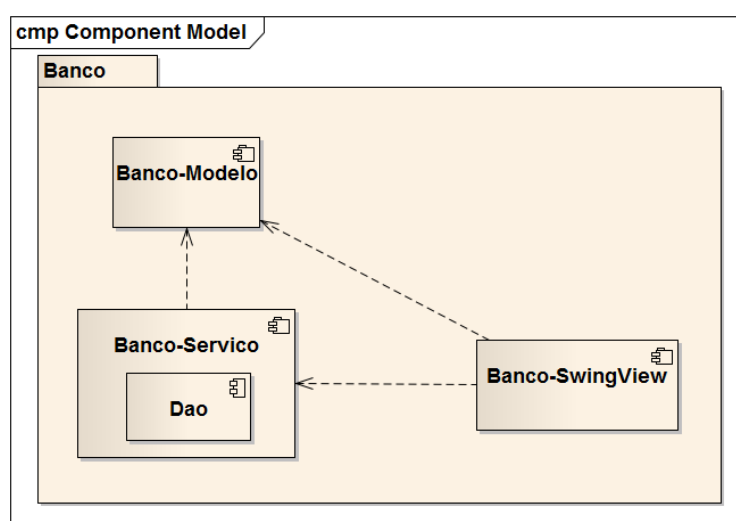


Figura 2 - Divisão de pacotes para aplicação Banco

1. Criação do projeto

A. Crie três projetos Java e os respectivos pacotes conforme a figura

3 abaixo:

- Banco-Modelo
- Banco-Servico
- Banco-SwingWeb

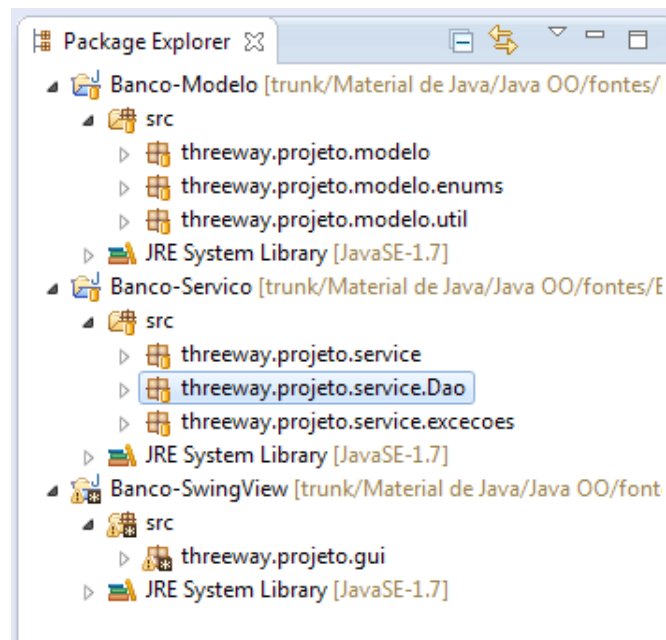


Figura 3 – Divisão de pacotes e criação de projeto para aplicação Banco

- B. Configure as dependências dos projetos. No build path do projeto Banco-Servico, na aba Projects adicione o projeto Banco-Modelo, e no projeto Banco-SwingView adicione Banco-Servico e Banco-Modelo.**
- C. Copie todas as classes que você utilizou nos laboratórios e distribua conforme a figura 4. Caso você não tenha desenvolvido anteriormente a classe, crie no pacote conforme a imagem abaixo.**

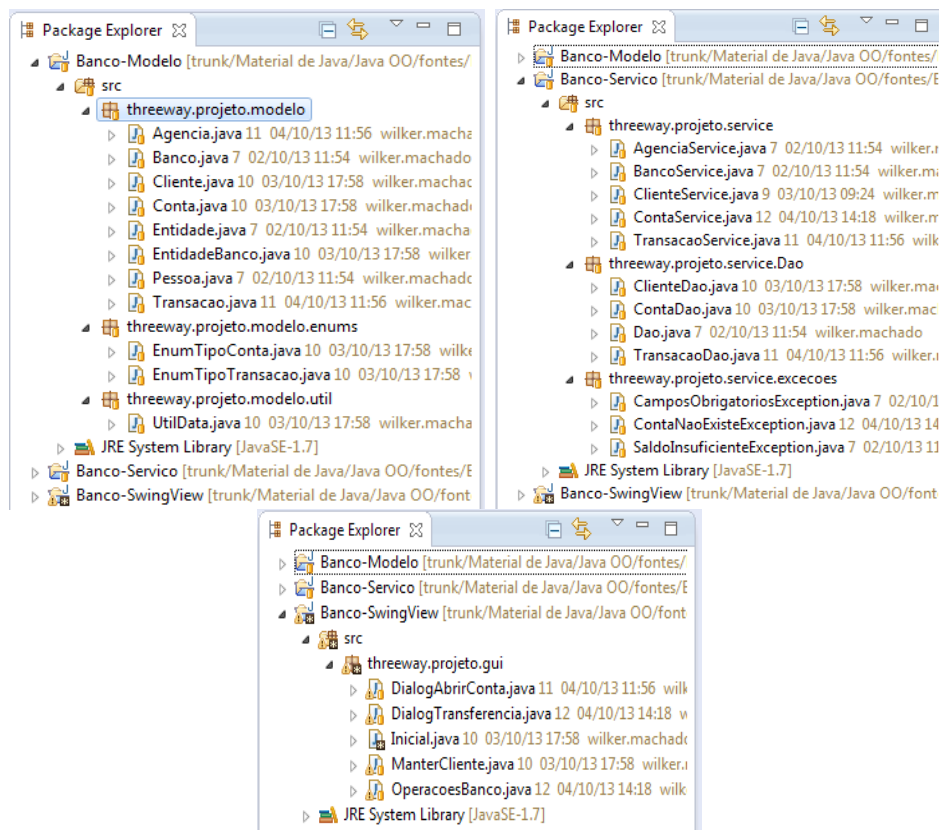


Figura 4 - Classes do projeto Banco

2. Prototipação - Criando as telas de acordo com os protótipos. Utilize o windowBuilder.

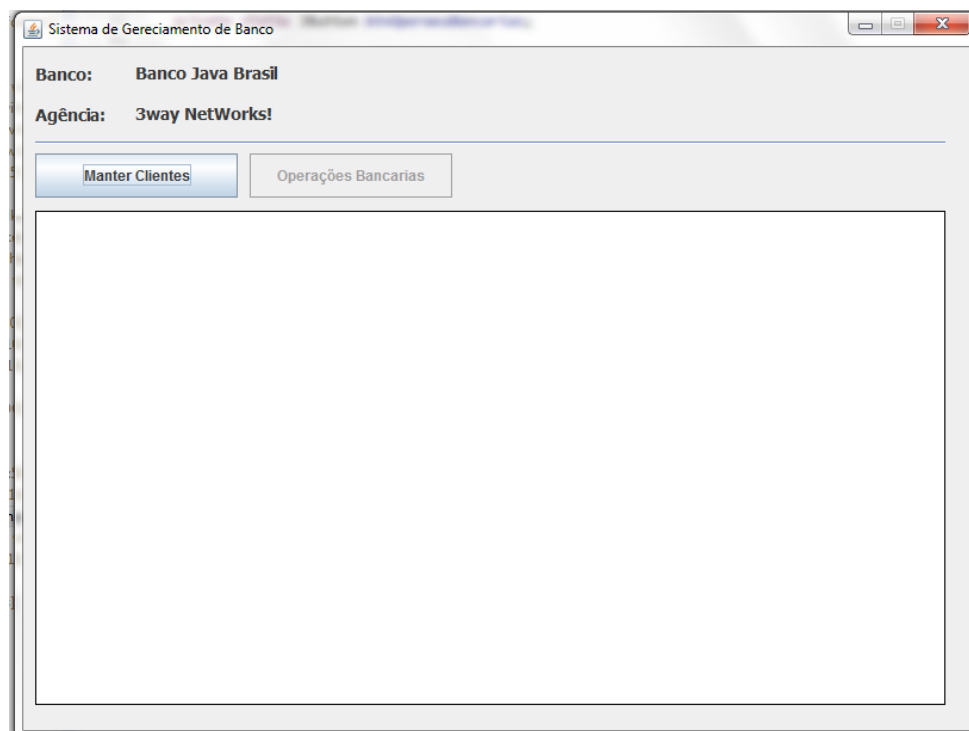
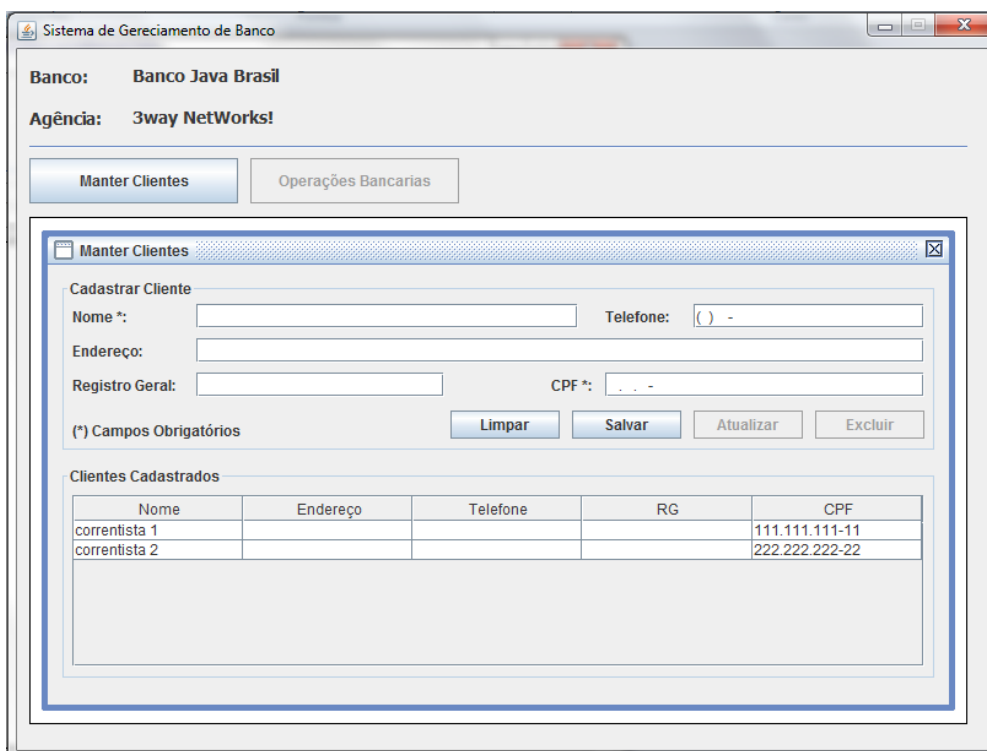


Figura 5 - Prototipação tela inicial (Frame)



Sistema de Gerenciamento de Banco

Banco: Banco Java Brasil

Agência: 3way NetWorks!

Manter Clientes Operações Bancárias

Manter Clientes

Cadastrar Cliente

Nome *: Telefone: () -

Endereço:

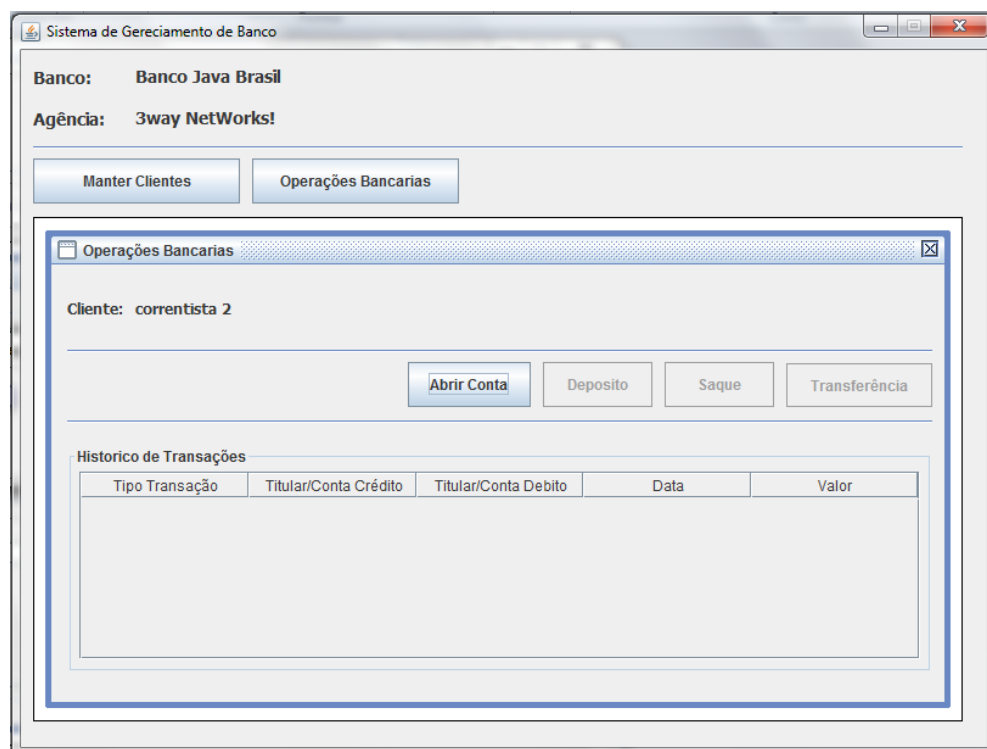
Registro Geral: CPF *:

(*) Campos Obrigatórios

Cientes Cadastrados

Nome	Endereço	Telefone	RG	CPF
correntista 1				111.111.111-11
correntista 2				222.222.222-22

Figura 6 - Prototipação tela Manter Clientes (JInternalFrame)



Sistema de Gerenciamento de Banco

Banco: Banco Java Brasil

Agência: 3way NetWorks!

Manter Clientes Operações Bancárias

Operações Bancárias

Cliente: correntista 2

Historico de Transações

Tipo Transação	Titular/Conta Crédito	Titular/Conta Debito	Data	Valor
----------------	-----------------------	----------------------	------	-------

Figura 7- Prototipação tela Operações Bancárias (JInternalFrame)

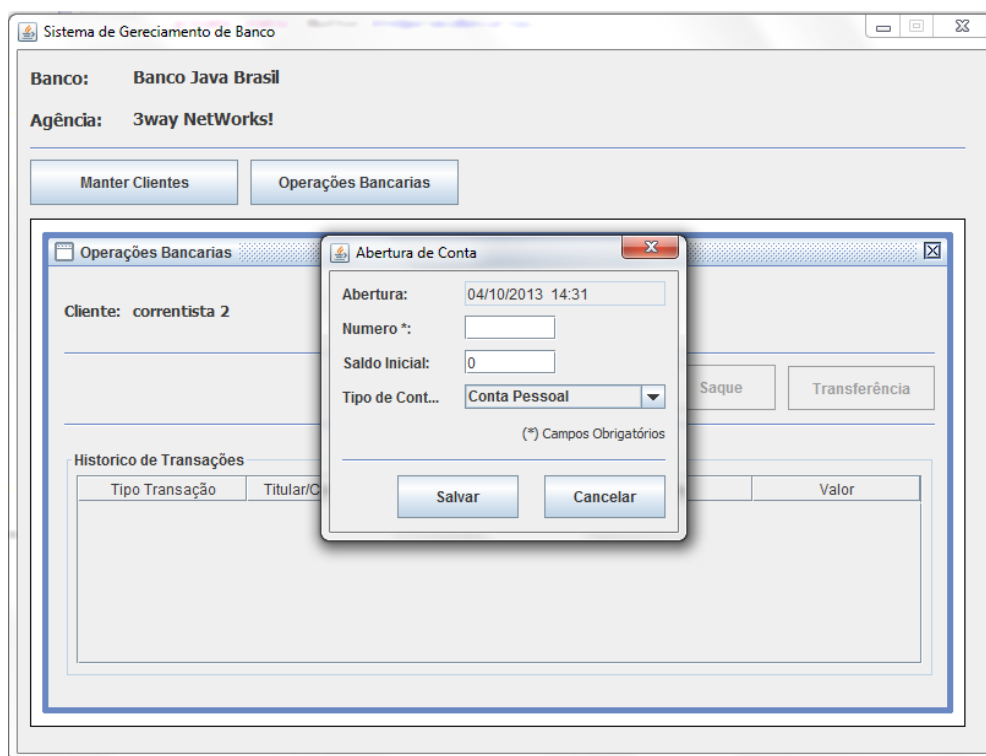


Figura 8- Prototipação tela Abrir Conta (JDialog)

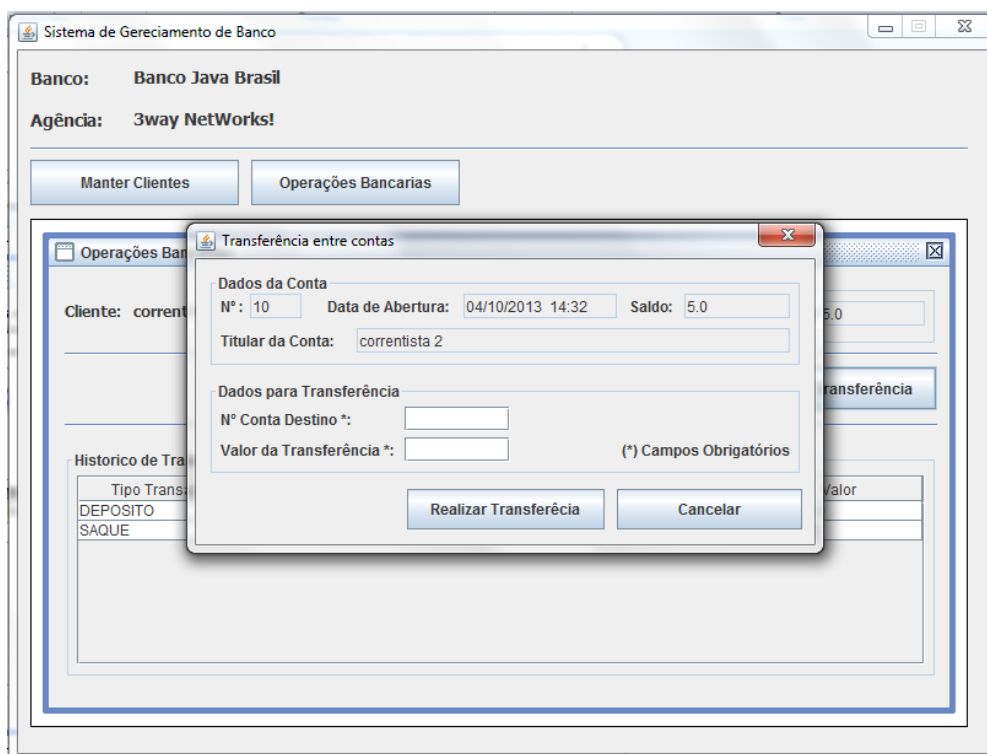


Figura 9- Prototipação tela Transferência entre contas (JDialog)

3. Ações de tela

A. Figura 5 - Tela inicial do sistema Banco

- 1.** Label com o nome do banco e agência será definida na classe AgenciaService.java
- 2.** Botão **Manter clientes** instanciará o JInternalFrame de Manter Clientes dentro de um JDesktopPane.
- 3.** Botão **Operações Bancárias** instanciará o JInternalFrame de Operações Bancárias dentro de um JDesktopPane. Manterá a instancia do cliente selecionado para ações do mesmo.
- 4.** Somente o botão de **Manter Clientes** estará habilitado, enquanto um cliente não estiver sido selecionado o botão de **Operações bancárias** estará desabilitado.

B. Figura 6 - Tela Manter Clientes (JInternalFrame)

- 1.** O botão (JButton) limpar terá função de limpar os campos de entrada (JTextField). Sempre estará habilitado.
- 2.** O botão de salvar terá função de salvar e validar os dados digitados pelo cliente. A validação vai ser a obrigatoriedade de inserir os valores de nome e CPF, caso estes não sejam inseridos o sistema deverá emitir uma mensagem avisando que os campos são de caráter obrigatório. Estará desabilitado quando um cliente for selecionado na lista.
- 3.** O botão atualizar terá função de validar e atualizar dados de um determinado cliente selecionado na lista. A validação vai ser a obrigatoriedade de inserir os valores de nome e CPF, caso estes não sejam inseridos o sistema deverá emitir uma mensagem avisando que os campos são de caráter obrigatório. Estará habilitado somente quando um cliente for selecionado.
- 4.** O botão de excluir terá função de excluir um determinado cliente selecionado na lista. Estará habilitado somente quando um cliente for selecionado.
- 5.** A tabela (JTable) listará todos os clientes cadastrados do sistema. Terá ação de MouseListener, ou seja quando houver click em algum cliente da tabela devesa ocorrer a ação de preenchimento dos campos de acordo com o cliente selecionado. Está função habilitará o botão de atualizar, excluir e operações bancárias e desabilitará o botão de salvar.

C. Figura 7 - Tela Operações Bancárias (JInternalFrame)

- 1.** Botão de abrir conta terá função de instanciar e mostrar a tela de abrir conta (JDialog). Caso já exista uma conta para o cliente selecionado o botão ficará desabilitado.

- 2.** Se houver alguma conta para o cliente, os botões de saque, depósito e transferência ficarão habilitados.
- 3.** Botão de Saque terá a ação de abrir um JOptionPane para o usuário informar o valor do saque que deseja fazer. Ao informar o valor será validado o formato que o usuário inseriu os dados e se há saldo suficiente para ação de saque, assim informando a mensagem de validação.
- 4.** Botão de Deposito terá a ação de abrir um JOptionPane para o usuário informar o valor do depósito que deseja fazer. . Ao informar o valor será validado o formato que o usuário inseriu os dados, assim informando a mensagem de validação.
- 5.** Botão de transferência terá função de instanciar e mostrar a tela de transferência (JDialog).
- 6.** A tabela (JTable) listará todas as transações feitas naquela conta, depósitos, saques e transferências.

D. Figura 8 - Tela Abrir Conta (JDialog)

- 1.** O botão cancelar terá ação de fechar a janela.
- 2.** O botão salvar terá a ação de validar e criar uma conta para o cliente. A validação vai ser a obrigatoriedade de inserir o numero da conta, caso este não seja inserido o sistema deverá emitir uma mensagem avisando que o campo é de caráter obrigatório.

E. Figura 8 - Tela Transferência (JDialog)

- 1.** O botão cancelar terá ação de fechar a janela.
- 2.** O botão realizar transferência terá a ação de validar e realizar a transferências entre as contas. A validação vai ser a obrigatoriedade dos campos numero de conta destino e valor da transferência, caso estes não sejam inseridos o sistema deverá emitir uma mensagem avisando que os campos são de caráter obrigatório. Outra validação será de numero de conta inexistente, ou seja, caso não exista uma conta com o numero informado o sistema deverá emitir uma mensagem avisando que não existe conta com esse numero. Outra validação será que o valor de, caso o valor informado não esta de acordo com o formato o sistema devera emitir uma mensagem informando o usuário a informar o valor corretamente.

4. Passo a Passo do código fonte do sistema

A. Dao - Data Access Object

1. Crie a interface Dao.java conforme a **listagem 1** abaixo, todas suas classes Dao implementarão essa interface. Lembre-se de sempre seguir a modelagem UML **figura 1** e a ordem de pacotes e projetos conforme a **figura 4** deste documento.

package threeway.projeto.service.Dao;

```
public interface Dao<E extends Entidade> {  
    /**  
     * Busca a entidade pelo seu identificador.  
     *  
     * @param identificador da entidade  
     * @return Entidade pesquisada  
     */  
    E obter(final Serializable identificador);  
  
    /**  
     * Altera a entidade.  
     *  
     * @param entidade  
     */  
    void alterar(final E entidade);  
  
    /**  
     * Insere a entidade.  
     *  
     * @param entidade  
     */  
    void salvar(final E entidade);  
  
    /**  
     * Remove a entidade.  
     *  
     * @param entidade  
     */  
    void remover(final E entidade);  
  
    /**  
     * Consulta os objetos que possuem os valores dos atributos do objeto informado.  
     *  
     * @param entidade que será utilizado como parâmetro na consulta.  
     * @return Collection<E>  
     */  
    Collection<E> consultar(final E entidade);  
  
    /**  
     * Lista todos os objetos da entidade.  
     *  
     * @return Collection<E>  
     */  
    Collection<E> listar();  
  
    /**  
     * Reordena todos os objetos da entidade.  
     *  
     * @return Collection<E>  
     */  
    void reordenar(final Collection<E> E);  
}
```

Listagem 1 - Dao.java

B. Figura 5 - Tela inicial do sistema Banco

1. Banco e Agência do sistema serão estáticos, ou seja será criada uma classe para cada um para criação de instancias estáticas para acesso de todo o

sistema. As classes serão **AgenciaService.java** e **BancoService.java**, siga a figura 4.

```
import threeway.projeto.modelo.Banco;

public class BancoService {

    public static final Banco bancoSistema() {
        Banco bancoSistema = new Banco();
        bancoSistema.setNome("Banco Java Brasil");
        return bancoSistema;
    }
}
```

Listagem 2 – BancoService.java

```
import threeway.projeto.modelo.Agencia;
import threeway.projeto.modelo.Banco;

public class AgenciaService {
    public static final Agencia agenciaSistema() {
        Banco bancoSistema = BancoService.bancoSistema();
        Agencia agenciaSistema = new Agencia("3way NetWorks!");
        agenciaSistema.setBanco(bancoSistema);
        bancoSistema.getAgencias().add(agenciaSistema);
        return agenciaSistema;
    }
}
```

Listagem 3 – AgenciaService.java

2. Crie a Classe **Inicial.java** pelo windowBuilder como um JFrame. Utilize o Design do window builder para criar o JFrame conforme o protótipo. Deixe os atributos necessários com visibilidade para toda a classe. Instancie **ClienteService.java** e **ContaService.java** pois estas instancias serão necessárias em outras telas, e assim podem ser passadas por parâmetro.

```
package threeway.projeto.gui;

public class Inicial {

    private ClienteService clienteService = new ClienteService();
    private ContaService contaService = new ContaService();
    private JFrame frmGereciamentoDeBanco;
    private JDesktopPane panelInterno;
    private JInternalFrame frameInterno;
    private JLabel lblBancoResult;
    private JLabel lblAgenciaResult;
    private JButton btnManterClientes;
    private JButton btnOperaesBancarias;
    private static Cliente clienteSelecionado;

} Listagem 4 – Inicial.java
```

3. Inicialize as variáveis pelo método initialize();

```
/**
 * Método responsável por inicializar todos os componentes swing deste frame
 */
private void initialize() {
    frmGereciamentoDeBanco = new JFrame();
    frmGereciamentoDeBanco.setResizable(false);
    frmGereciamentoDeBanco.setTitle("Sistema de Gereciamento de Banco");
    frmGereciamentoDeBanco.setBounds(100, 100, 800, 600);
    frmGereciamentoDeBanco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frmGereciamentoDeBanco.getContentPane().setLayout(null);
    frmGereciamentoDeBanco.setLocationRelativeTo(null);
}
```

```

JLabel lblBanco = new JLabel("Banco:");
lblBanco.setFont(new Font("Tahoma", Font.BOLD, 14));
lblBanco.setBounds(10, 11, 74, 23);
frmGereciamentoDeBanco.getContentPane().add(lblBanco);

JLabel lblAgencia = new JLabel("Agência:");
lblAgencia.setFont(new Font("Tahoma", Font.BOLD, 14));
lblAgencia.setBounds(10, 45, 74, 23);
frmGereciamentoDeBanco.getContentPane().add(lblAgencia);

btnManterClientes = new JButton("Manter Clientes");
btnManterClientes.setBounds(10, 89, 170, 37);
frmGereciamentoDeBanco.getContentPane().add(btnManterClientes);

btnOperaesBancarias = new JButton("Operações Bancarias");
btnOperaesBancarias.setBounds(190, 89, 170, 37);
btnOperaesBancarias.setEnabled(Boolean.FALSE);
frmGereciamentoDeBanco.getContentPane().add(btnOperaesBancarias);

lblBancoResult = new JLabel("...");
lblBancoResult.setFont(new Font("Tahoma", Font.BOLD, 14));
lblBancoResult.setBounds(94, 11, 480, 20);
frmGereciamentoDeBanco.getContentPane().add(lblBancoResult);

lblAgenciaResult = new JLabel("...");
lblAgenciaResult.setFont(new Font("Tahoma", Font.BOLD, 14));
lblAgenciaResult.setBounds(94, 45, 480, 20);
frmGereciamentoDeBanco.getContentPane().add(lblAgenciaResult);

JSeparator separator = new JSeparator();
separator.setBounds(10, 79, 764, 11);
frmGereciamentoDeBanco.getContentPane().add(separator);

panelInterno = new JPanel();
panelInterno.setBorder(new MatteBorder(1, 1, 1, 1, (Color) new Color(0, 0, 0)));
panelInterno.setBounds(10, 137, 764, 414);
frmGereciamentoDeBanco.getContentPane().add(panelInterno);
panelInterno.setLayout(null);

frameInterno = new JFrame("New JFrame");
frameInterno.setBounds(10, 11, 744, 392);
panelInterno.add(frameInterno);
frameInterno.setVisible(false);
}

```

Listagem 6 - Inicial.java - método initialize();

4. No construtor da classe **Inicial.java** chame o método **initialize()** e preencha os label com o resultado da **listagem 3** informando a agência e banco do sistema. Acessando o método estático da classe **AgenciaService.java**.

```

/**
 * Responsável pela criação de novas instâncias desta classe.
 */
public Inicial() {
    initialize();
    lblAgenciaResult.setText(AgenciaService.agenciaSistema().getNome());
    lblBancoResult.setText(AgenciaService.agenciaSistema().getBanco().getNome());
    eventoDosBotoes(); // método que adiciona evento aos botões
}

```

Listagem 6 - Inicial.java - Construtor

5. Crie o método **eventoDosBotoes()**, onde será instanciado os eventos dos botões da tela.

```

/**
 * Método responsável por adicionar evento aos botões
 */

```

```

*/
public void eventoDosBotoes() {
    btnOperacoesBancarias.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            mostraTelaOperacoesBancarias();
        }
    });

    btnManterClientes.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            mostraTelaManterClientes();
        }
    });
}

```

Listagem 7 - Inicial.java - eventoDosBotoes()

6. Foi criado os métodos **mostraTelaOperacoesBancarias()** e **mostraTelaManterClientes()**, isso para deixar o código mais organizado e diminuindo a complexidade dos métodos.

```

/**
 * Método responsável por instanciar e mostrar a tela de Manter Clientes
 */
public void mostraTelaManterClientes() {
    try {
        JInternalFrame newFrame = new ManterCliente(clienteService);
        newFrame.setBounds(frameInterno.getBounds());
        panelInterno.add(newFrame);
        newFrame.show();
    } catch (ParseException e1) {
        e1.printStackTrace();
    }
}

/**
 * Método responsável por instanciar e mostrar a tela de Operações Bancarias
 */
public void mostraTelaOperacoesBancarias() {
    JInternalFrame newFrame = new OperacoesBanco(this.contaService,
clienteSelecionado);
    newFrame.setBounds(frameInterno.getBounds());
    panelInterno.add(newFrame);
    newFrame.show();
}

```

Listagem 8 - Inicial.java - mostraTelaOperacoesBancarias() e mostraTelaManterClientes()

7. Repare que não existe os construtores para **OperacoesBanco()** e **ManterCliente()**, iremos criar estes a seguir.

8. Crie o método estático para mudar habilitar o botão operações bancarias e setar o objeto cliente para tornar possíveis as ações da tela de operações bancarias.

C. Figura 6 - Tela Manter Clientes

1. Crie a Classe **ClienteDao.java** implementando a interface **Dao<?>**. Siga a **listagem 9** abaixo.

```
package threeway.projeto.service.Dao;
```

```
public class ClienteDao implements Dao<Cliente> {

    private Collection<Cliente> clientesMemoria = new ArrayList<Cliente>();

    @Override
    public Cliente obter(Serializable identificador) {
        return null;
    }

    @Override
    public void alterar(Cliente entidade) {
        for (Cliente cliente : clientesMemoria) {
            System.out.println("Buscando...");
            System.out.println("Ent CPF = "+entidade.getCpf()+" Cl CPF = "+cliente.getCpf());

            if (entidade.getCpf().equals(cliente.getCpf())) {
                clientesMemoria.remove(cliente);
                clientesMemoria.add(entidade);
                System.out.println("achou e removeu");
                break;
            }
        }
    }

    @Override
    public void salvar(Cliente entidade) {
        this.clientesMemoria.add(entidade);
    }

    @Override
    public void remover(Cliente entidade) {
        this.clientesMemoria.remove(entidade);
    }

    @Override
    public Collection<Cliente> consultar(Cliente entidade) {
        return null;
    }

    @Override
    public Collection<Cliente> listar() {
        return this.clientesMemoria;
    }

    @Override
    public void reordenar(Collection<Cliente> E) {
        clientesMemoria = E;
    }
}
```

Listagem 9 - ClienteDao.java

2. Crie a Classe **ClienteService.java** instanciando um objeto do tipo **ClienteDao** para acesso aos dados em memória. Siga a **listagem 10** abaixo.

```
package threeway.projeto.service;
```

```
public class ClienteService {

    private ClienteDao dao = new ClienteDao();

    /**
     * Método responsável por Atualizar entidade cliente
     */
}
```

```

    *
    * @param cliente
    *
    * @throws CamposObrigatoriosException
    */
    public void atualizar(Cliente cliente) throws CamposObrigatoriosException ,
        this.validarCamposObrigatorios(cliente);
        this.getDao().alterar(cliente);
    }

    /**
    * Método responsável por excluir entidade cliente
    *
    * @param cliente
    */
    public void excluir(Cliente cliente) {
        this.getDao().remove(cliente);
    }

    /**
    * Método responsável por listar todos os clientes
    *
    * @return ArrayList<Cliente>
    */
    public ArrayList<Cliente> listarTodosClientes() {
        return new ArrayList<Cliente>(this.getDao().listar());
    }

    /**
    * Método responsável por salvar a entidade cliente
    *
    * @param cliente
    *
    * @throws CamposObrigatoriosException
    */
    public void salvar(Cliente cliente) throws CamposObrigatoriosException,
        CPFJaCadastradoException, RgJaCadastradoException{
        this.validarCamposObrigatorios(cliente);
        this.validarCpfRepetido(cliente);
        this.validarRgRepetido(cliente);
        this.getDao().salvar(cliente);
    }

    /**
    * Método responsável por validar campos obrigatorios Nome e CPF de cliente
    *
    * @param cliente
    *
    * @throws CamposObrigatoriosException
    */
    private void validarCamposObrigatorios(Cliente cliente) throws
        CamposObrigatoriosException {
        if (cliente == null || cliente.getNome().equals("") ||
            cliente.getCpf().replace("-", "").replace(".", "").trim().equals("")) {
            throw new CamposObrigatoriosException();
        }
    }

    private void validarCpfRepetido(Cliente cliente) throws
        CPFJaCadastradoException{
        for(Cliente entidade : getDao().listar()){
            if(entidade.getCpf().equals(cliente.getCpf())){
                throw new CPFJaCadastradoException();
            }
        }
    }

    private void validarRgRepetido(Cliente cliente) throws
        RgJaCadastradoException{
        for(Cliente entidade : getDao().listar()){
            if(entidade.getRg().equals(cliente.getRg())){
                throw new RgJaCadastradoException();
            }
        }
    }

```

```
    }

    /**
     * Método responsável por retornar a instancia de ClienteDao
     *
     * @return
     */
    public ClienteDao getDao() {
        return dao;
    }

    /**
     * Método responsável por reordenar a Tabela
     *
     * @return
     */
    public void reordenaTable(){
        getDao().reordenar(getDao().listar());
    }
}

Listagem 10 - ClienteService.java
```

3. Veja que você precisará implementar a classe **CamposObrigatoriosException.java** para tratar a exceção quando o usuário não informar os campos obrigatórios definidos. Siga a **listagem 11** abaixo.

```
package threeway.projeto.service.excecoes;

public class CamposObrigatoriosException extends Exception {

    private static final long serialVersionUID = -5487345156293002556L;

    /**
     * Responsável pela criação de novas instâncias desta classe.
     */
    public CamposObrigatoriosException() {

        super("Por Favor preencha todos campos obrigatórios!");
    }

    /**
     * Responsável pela criação de novas instâncias desta classe.
     *
     * @param mensagem
     */
    public CamposObrigatoriosException( String mensagem ) {

        super(mensagem);
    }

}

Listagem 11 - CamposObrigatoriosException.java
```

4. Crie a Classe **ManterCliente.java** pelo windowBuilder como um JInternalFrame. Utilize o Design do window builder para criar conforme o protótipo. Deixe os atributos necessários com visibilidade para toda a classe.

```
package threeway.projeto.gui;

private ClienteService service;
private JTextField tfNome;
private JTextField tfEndereco;
private JTextField tfRg;
private JFormattedTextField ftfCpf;
private JFormattedTextField ftfTelefone;
private JTable tableClientes;
private JButton btnSalvar;
```

```
private JButton btnAtualizar;
private JButton btnExcluir;
private JButton btnLimpar;
private JPanel panelLista;
private Boolean clienteSelecioneado = Boolean.FALSE;
private Cliente cliente = new Cliente();
}
```

Listagem 12 – ManterCliente.java

5. Inicialize as variáveis pelo método initialize(). Siga a **listagem 13** abaixo.

```
/**
 * Método responsável por inicializar componentes da tela
 *
 * @throws ParseException
 */
private void initialize() throws ParseException {

    setClosable(true);
    setTitle("Manter Clientes");

    setBounds(100, 100, 750, 375);
    getContentPane().setLayout(null);

    JPanel panelCad = new JPanel();

    panelCad.setBorder(new
TitledBorder(UIManager.getBorder("TitledBorder.border"), "Cadastrar Cliente", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
    panelCad.setBounds(10, 11, 714, 142);
    getContentPane().add(panelCad);
    panelCad.setLayout(null);

    JLabel lblNome = new JLabel("Nome *");
    lblNome.setBounds(10, 24, 91, 14);
    panelCad.add(lblNome);

    JLabel lblTelefone = new JLabel("Telefone:");
    lblTelefone.setBounds(444, 24, 62, 14);
    panelCad.add(lblTelefone);

    JLabel lblEndereco = new JLabel("Endereço:");
    lblEndereco.setBounds(10, 52, 91, 14);
    panelCad.add(lblEndereco);

    JLabel lblRegistroGeral = new JLabel("Registro Geral *");
    lblRegistroGeral.setBounds(10, 80, 91, 14);
    panelCad.add(lblRegistroGeral);

    JLabel lblCpf = new JLabel("CPF *");
    lblCpf.setBounds(400, 80, 34, 14);
    panelCad.add(lblCpf);

    tfNome = new JTextField();
    tfNome.setBounds(111, 21, 311, 20);
    panelCad.add(tfNome);
    tfNome.setColumns(10);

    ftfTelefone = new JFormattedTextField(new MaskFormatter("(##)####-####"));
    ftfTelefone.setBounds(516, 21, 188, 20);
    panelCad.add(ftfTelefone);

    tfEndereco = new JTextField();
    tfEndereco.setBounds(111, 49, 593, 20);
    panelCad.add(tfEndereco);
    tfEndereco.setColumns(10);

    tfRg = new JTextField();
    tfRg.setBounds(111, 77, 202, 20);
    panelCad.add(tfRg);

    tfRg.setColumns(10);
    ftfCpf = new JFormattedTextField(new MaskFormatter("###.###.###-##"));
```



```

ftfCpf.setBounds(444, 77, 260, 20);
panelCad.add(ftfCpf);
ftfCpf.setColumns(10);

btnSalvar = new JButton("Salvar");
btnSalvar.setBounds(417, 108, 89, 23);
panelCad.add(btnSalvar);

btnAtualizar = new JButton("Atualizar");
btnAtualizar.setBounds(516, 108, 89, 23);
panelCad.add(btnAtualizar);

btnExcluir = new JButton("Excluir");
btnExcluir.setBounds(615, 108, 89, 23);
panelCad.add(btnExcluir);

JLabel lblCamposObrigatorios = new JLabel("(*) Campos Obrigatórios");
lblCamposObrigatorios.setBounds(10, 117, 174, 14);
panelCad.add(lblCamposObrigatorios);

btnLimpar = new JButton("Limpar");
btnLimpar.setBounds(318, 108, 89, 23);
panelCad.add(btnLimpar);

panelLista = new JPanel();
panelLista.setBorder(new
TitledBorder(UIManager.getBorder("TitledBorder.border"), "Clientes Cadastrados", TitledBorder.LEADING,
TitledBorder.TOP, null, null));
panelLista.setBounds(10, 164, 714, 175);
getContentPane().add(panelLista);
panelLista.setLayout(null);
}

Listagem 13 - ManterCliente.java - initialize()

```

6. No construtor da classe **ManterCliente.java** chame o método **initialize()** e faça com que receba uma instancia de **ClienteService.java**.

```

/**
 * Responsável pela criação de novas instâncias desta classe.
 *
 * @param service
 *
 * @throws ParseException
 */
public ManterCliente( ClienteService service ) throws ParseException {

    this.service = service;
    initialize();
    defineEnabledBotoes();
    iniciaJTable();
    atualizaDadosTabela();
    addEventoMouseListenerTabela();
    acaoBotaoSalvar();
    acaoBotaoLimpar();
    acaoBotaoExcluir();
    acaoBotaoAtualizar();

}

Listagem 14 - ManterCliente.java -construtor()

```

7. Crie os métodos que estão sendo chamados no construtor. Siga a **listagem 15** abaixo.

```

/**
 * Método responsável por definir quando os botoes estarão habilitados
 *
 */

```

```

private void defineEnabledBotoes() {
    btnExcluir.setEnabled(clienteSelecioneado);
    btnAtualizar.setEnabled(clienteSelecioneado);
    btnSalvar.setEnabled(!clienteSelecioneado);
    ftfCpf.setEditable(!clienteSelecioneado);
    tfRg.setEditable(!clienteSelecioneado);
    Inicial.modificaBotaoOperacoes(clienteSelecioneado, cliente);
}

/**
 * Método responsável por adicionar evento de click na tabela de clientes
 */
private void addEventoMouseListenerTabela() {
    tableClientes.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            int l = tableClientes.getSelectedRow();
            cliente = service.listarTodosClientes().get(l);
            carregarValores(cliente);
        }
    });
}

/**
 * Método responsável por carregar os valores de cliente
 * @param cliente
 */
public void carregarValores(Cliente cliente) {
    ftfCpf.setText(cliente.getCpf());
    ftfTelefone.setText(cliente.getTelefone());
    tfEndereco.setText(cliente.getEndereco());
    tfNome.setText(cliente.getNome());
    tfRg.setText(cliente.getRg());
    this.clienteSelecioneado = Boolean.TRUE;
    defineEnabledBotoes();
}

/**
 * Método responsável por inicializar componente JTable
 */
private void iniciaJTable() {
    tableClientes = new JTable();
    tableClientes.setModel(new DefaultTableModel(new Object[][] {},
        retornaNomeColunas()));
    tableClientes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    tableClientes.setBounds(703, 24, -690, 140);
    tableClientes.getTableHeader().setReorderingAllowed(false);
    JScrollPane scrollPane = new JScrollPane(tableClientes);
    scrollPane.setViewportView(tableClientes);
    scrollPane.setBounds(10, 24, 694, 140);
    panelLista.add(scrollPane);
}

/**
 * Método responsável por atualizar dados da JTable
 */
private void atualizaDadosTabela() {
    tableClientes.setModel(new DefaultTableModel(new Object[][] {
        {}, retornaNomeColunas()
    }));
    tableClientes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    DefaultTableModel dtm = (DefaultTableModel) tableClientes.getModel();
    Cliente cli = new Cliente();
    ArrayList<Cliente> listaCliente = service.listarTodosClientes();
    Collections.sort(listaCliente);
    Iterator<Cliente> it = listaCliente.iterator();
    while (it.hasNext()) {
        cli = it.next();
        dtm.addRow(new Object[] { cli.getNome(), cli.getEndereco(),
            cli.getTelefone(),
            cli.getRg(), cli.getCpf() });
    }
    service.getDao().reordenar(listaCliente);
}

```

```

    }

    /**
     * Método responsável por retornaqr o nome das colunas da tabela clientes
     *
     * @return String[]
     */
    private String[] retornaNomeColunas() {
        String[] columnNames = new String[5];
        columnNames[0] = ( "Nome" );
        columnNames[1] = ( "Endereço" );
        columnNames[2] = ( "Telefone" );
        columnNames[3] = ( "RG" );
        columnNames[4] = ( "CPF" );
        return columnNames;
    }

    /**
     * Método responsável por adicionar ação ao botão limpar
     *
     */
    private void acaoBotaoLimpar() {
        btnLimpar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                limpar();
                clienteSelecionado = Boolean.FALSE;
                defineEnabledBotoes();
            }
        });
    }

    /**
     * Método responsável por adicionar ação ao botão salvar
     *
     */
    private void acaoBotaoSalvar() {
        btnSalvar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                try {
                    preencheClienteComCampos();
                    service.salvar(cliente);
                    JOptionPane.showMessageDialog(null, "Cliente salvo com
sucesso!");

                    limpar();
                } catch (CamposObrigatoriosException ex) {
                    JOptionPane.showMessageDialog(null, ex.getMessage());
                } catch (CPFJaCadastradoException ex) {
                    JOptionPane.showMessageDialog(null, ex.getMessage());
                } catch (RgJaCadastradoException ex) {
                    JOptionPane.showMessageDialog(null, ex.getMessage());
                } finally {
                    atualizaDadosTabela();
                }
            }
        });
    }

    /**
     * Método responsável por adicionar ação ao botão excluir
     *
     */
    private void acaoBotaoExcluir() {
        btnExcluir.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                service.excluir(cliente);
                limpar();
                clienteSelecionado = Boolean.FALSE;
                defineEnabledBotoes();
                atualizaDadosTabela();
                JOptionPane.showMessageDialog(null, "Cliente removido com
sucesso!");
            }
        });
    }
}

```

```

/**
 * Método responsável por adicionar ação ao botão atualizar
 */
private void acaoBotaoAtualizar() {
    btnAtualizar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                preencheClienteComCampos();
                service.atualizar(cliente);
                service.reordenaTable();
                limpar();
                clienteSelecioneado = Boolean.FALSE;
                defineEnabledBotoes();
                JOptionPane.showMessageDialog(null, "Cliente atualizado
com sucesso!");
            } catch (CamposObrigatoriosException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage());
            } catch (RgJaCadastradoException ex) {
                JOptionPane.showMessageDialog(null, ex.getMessage());
            } finally {
                atualizaDadosTabela();
            }
        }
    });
}

/**
 * Método responsável por recuperar dados do JTextField e setar em cliente
 */
public void preencheClienteComCampos() {
    cliente = new Cliente();
    cliente.setNome(tfNome.getText());
    cliente.setEndereco(tfEndereco.getText());
    cliente.setCpf(ftfCpf.getText());
    cliente.setRg(tfrg.getText());
    cliente.setTelefone(ftfTelefone.getText());
}

/**
 * Método responsável por limpar campos JTextField
 */
public void limpar() {
    ftfCpf.setText("");
    ftfTelefone.setText("");
    tfEndereco.setText("");
    tfNome.setText("");
    tfrg.setText("");
    cliente = new Cliente();
}

```

Listagem 15 - ManterCliente.java –métodos da classe

8. Veja que você precisará implementar a classe **CPFJaCadastradoException.java** para tratar a exceção quando o usuário informar um CPF que já foi cadastrado. Siga a **listagem 16** abaixo.

package threeway.projeto.service.excecoes;

```

public class CPFJaCadastradoException extends Exception{

    public CPFJaCadastradoException() {
        super("CPF ja cadastrado.");
    }

    public CPFJaCadastradoException(String mensagem) {
        super(mensagem);
    }
}

```

```
}
```

Listagem 16 – CPFJaCadastradoException.java – classe

9. Veja que você precisará implementar a classe **RGJaCadastradoException.java** para tratar a exceção quando o usuário informar um RG que já foi cadastrado. Siga a **listagem 17** abaixo.

```
package threeway.projeto.service.excecoes;
```

```
public class RgJaCadastradoException extends Exception{

    public RgJaCadastradoException() {
        super("RG ja cadastrado.");
    }

    public RgJaCadastradoException(String mensagem) {
        super(mensagem);
    }

}
```

Listagem 17 – RGJaCadastradoException.java - classe

D. Figura 7 - Tela Operações Bancárias

1. Crie a Classe **ContaDao.java** implementando a interface **Dao<?>**. Siga a **listagem 18** abaixo. Observe que os métodos que não serão utilizados não foi implementado, pois os requisitos do sistema não contemplam tais ações.

```
package threeway.projeto.service.Dao;
```

```
public class ContaDao implements Dao<Conta> {

    private Collection<Conta> contasMemoria = new ArrayList<Conta>();

    @Override
    public Conta obter(Serializable identificador) {
        return null;
    }

    @Override
    public void alterar(Conta entidade) {}

    @Override
    public void salvar(Conta entidade) {
        contasMemoria.add(entidade);
    }

    @Override
    public void remover(Conta entidade) {}

    @Override
    public Collection<Conta> consultar(Conta entidade) {
        return null;
    }

    @Override
    public Collection<Conta> listar() {
        return contasMemoria;
    }

    @Override
    public void reordenar(Collection<Conta> E) {
        contasMemoria = E;
    }

}
```

Listagem 18 – ContaDAO.java

2. Modifique a Classe **ContaService.java** que veio sendo modificada ao longo dos laboratórios realizados, instanciando um objeto do tipo **ContaDao** para acesso aos dados em memória e **TransacaoService** para manipular ações direcionadas as transações efetuadas. Siga a **listagem 19** abaixo.

```
package threeway.projeto.service;
```

```
public class ContaService {
```

```
    private ContaDao dao = new ContaDao();
```

```
    private TransacaoService transacaoService = new TransacaoService();
```

```
    /**
```

```
     * Método responsável por realizar ação de depósito
```

```
     *
```

```
     * @param contaDestino
```

```
     *
```

```
     * @param valor
```

```
    */
```

```
    public void depositar(Conta contaDestino, double valor) {
        contaDestino.setSaldo(contaDestino.getSaldo() + valor);
```

```
        this.historicoTransacao(null, contaDestino, valor, "depósito na conta "
```

```
        contaDestino.getNumero(), EnumTipoTransacao.DEPOSITO);
```

```
    }
```

```
    /**
```

```
     * Método responsável por realizar ação de saque caso não tenha saldo
```

```
     *
```

```
     * lançado uma exceção
```

```
     *
```

```
     * @param contaSaque
```

```
     *
```

```
     * @param valor
```

```
     *
```

```
     * @throws SaldoInsuficienteException
```

```
    */
```

```
    public void sacar(Conta contaSaque, double valor) throws
```

```
        if (contaSaque.getSaldo() - valor >= 0) {
```

```
            contaSaque.setSaldo(contaSaque.getSaldo() - valor);
```

```
            this.historicoTransacao(null, contaSaque, valor, "saque na
```

```
            contaSaque.getNumero(), EnumTipoTransacao.SAQUE);
```

```
        } else {
```

```
            throw new SaldoInsuficienteException();
```

```
        }
```

```
    }
```

```
    /**
```

```
     * Método responsável por realizar transferencia entre contas caso não tenha
```

```
     *
```

```
     * suficiente e lançado uma exceção
```

```
     *
```

```
     * @param contaSaque
```

```
     *
```

```
     * @param valor
```

```
     *
```

```
     * @param contaDestino
```

```
     *
```

```
     * @return boolean true se a transferencia for realizada com sucesso
```

```
     *
```

```
     * @throws SaldoInsuficienteException
```

```
    */
```

```
    public boolean transferir(Conta contaSaque, double valor, Conta contaDestino)
```

```
    throws SaldoInsuficienteException {
```

```
        return transferir(contaSaque, valor, contaDestino, "transferencia para
```

```
        conta " + contaDestino.getNumero());
```

```
    }
```

saldo

```
/**
 * Método responsável por realizar transferencia entre contas caso não tenha
 * suficiente e lançado uma exceção
 */
@param contaSaque
@param valor
@param contaDestino
@param descr
@return boolean true se a transferencia for realizada com sucesso
@throws SaldoInsuficienteException
*/
public boolean transferir(Conta contaSaque, double valor, Conta contaDestino,
String descr) throws SaldoInsuficienteException {
    if (contaSaque.getSaldo() - valor >= 0) {
        this.debito(contaSaque, valor);
        this.credito(contaDestino, valor);
        this.historicoTransacao(contaSaque, contaDestino, valor, descr,
EnumTipoTransacao.TRANSFERENCIA);
        return true;
    } else {
        throw new SaldoInsuficienteException();
    }
}

/**
 * Método responsável por realizar debitos na conta
 */
@param contaOperacao
@param valor
*/
protected void debito(Conta contaOperacao, double valor) {
    contaOperacao.setSaldo(contaOperacao.getSaldo() - valor);
}

/**
 * Método responsável por realizar operações de credito na conta
 */
@param contaOperacao
@param valor
*/
protected void credito(Conta contaOperacao, double valor) {
    contaOperacao.setSaldo(contaOperacao.getSaldo() + valor);
}

/**
 * Método responsável por instanciar e gravar historico de transações
 */
@param contaDebito
@param contaCredito
@param valor
@param descr
@param tipoTransacao
*/
protected void historicoTransacao(Conta contaDebito, Conta contaCredito,
double valor, String descr, EnumTipoTransacao tipoTransacao) {
    Transacao transacao = new Transacao(UtilData.data(), contaDebito,
contaCredito, valor, descr, tipoTransacao);
    if (contaDebito != null) {
        contaDebito.getTransacoes().add(transacao);
    }
    contaCredito.getTransacoes().add(transacao);
}
```

```

        transacaoService.salvar(transacao);
    }

    /**
     * Método responsável por salvar entidade conta validando campos obrigatorios
     *
     * @param conta
     *
     * @throws CamposObrigatoriosException
     */
    public void salvar(Conta conta) throws CamposObrigatoriosException,
NumeroDeContaJaExistenteException {
        this.validarCamposObrigatorios(conta);
        this.validarNumeroDeConta(conta);
        this.getDao().salvar(conta);
    }

    private void validarNumeroDeConta(Conta conta) throws
NumeroDeContaJaExistenteException{
        for(Conta entidade : getDao().listar()){
            if(entidade.getNumero().equals(conta.getNumero())){
                throw new NumeroDeContaJaExistenteException();
            }
        }
    }

    /**
     * Método responsável por validar campos obrigatorios
     *
     * @param conta
     *
     * @throws CamposObrigatoriosException
     */
    private void validarCamposObrigatorios(Conta conta) throws
CamposObrigatoriosException{
        if (conta == null || conta.getTitular() == null || conta.getNumero() ==
0) {
            throw new CamposObrigatoriosException();
        }
    }

    /**
     * Método responsável por buscar conta pelo cliente
     *
     * @param cliente
     *
     * @return Conta
     */
    public Conta buscaContaPorCliente(Cliente cliente) {
        for (Conta conta : getDao().listar()) {
            if (conta.getTitular().equals(cliente)) {
                return conta;
            }
        }
        return null;
    }

    /**
     * Método responsável por buscar conta por numero caso nao exista lança exceção
     */
    ContaNaoExisteException
    {
        @param numeroConta
        @return Conta
        @throws ContaNaoExisteException
    }
    public Conta buscaContaPorNumero(int numeroConta) throws
ContaNaoExisteException {
        for (Conta conta : getDao().listar()) {
            if (conta.getNumero() == numeroConta) {
                return conta;
            }
        }
    }

```



```
        }  
        throw new ContaNaoExisteException();  
    }  
  
    /**  
     * Método responsável por retornar instancia de ContaDao  
     *  
     * @return  
     */  
    public ContaDao getDao() {  
        return dao;  
    }  
}
```

Listagem 19 - ContaService.java

3. Crie a Classe **TransacaoService.java** e **TransacaoDao.java** conforme a listagem 20 e 21 respectivamente.

```
package threeway.projeto.service.Dao;  
  
public class TransacaoDao implements Dao<Transacao> {  
  
    private Collection<Transacao> transacoesMemoria = new ArrayList<Transacao>();  
  
    @Override  
    public Transacao obter(Serializable identificador) {  
        return null;  
    }  
  
    @Override  
    public void alterar(Transacao entidade) {}  
  
    @Override  
    public void salvar(Transacao entidade) {  
        this.transacoesMemoria.add(entidade);  
    }  
  
    @Override  
    public void remover(Transacao entidade) {}  
  
    @Override  
    public Collection<Transacao> consultar(Transacao entidade) {  
        return null;  
    }  
  
    @Override  
    public Collection<Transacao> listar() {  
        return transacoesMemoria;  
    }  
  
    @Override  
    public void reordenar(Collection<Transacao> E) {  
        transacoesMemoria = E;  
    }  
}
```

Listagem 20 - TransacaoDao.java

```
package threeway.projeto.service;  
  
public class TransacaoService {  
  
    TransacaoDao dao = new TransacaoDao();  
  
    /**  
     * Método responsável por retornar instancia de TransacaoDao  
     *  
     * @return TransacaoDao  
     */  
    public TransacaoDao getDao() {  
        return dao;  
    }  
}
```

```
    }

    /**
     * Método responsável por salvar a entidade transação
     *
     * @param transacao
     */
    public void salvar(Transacao transacao) {
        this.getDao().salvar(transacao);
    }
}

Listagem 21 – TransacaoService.java
```

4. Veja que você precisará implementar a classe **ContaNaoExisteException.java** para tratar a exceção quando o usuário informar um numero de conta que não existe. Siga a **listagem 22** abaixo.

```
package threeway.projeto.service.excecoes;

public class ContaNaoExisteException extends Exception{

    private static final long serialVersionUID = -2692264935862218672L;

    /**
     * Responsável pela criação de novas instâncias desta classe.
     */
    public ContaNaoExisteException() {

        super("Conta não existe.");
    }

    /**
     * Responsável pela criação de novas instâncias desta classe.
     *
     * @param mensagem
     */
    public ContaNaoExisteException(String mensagem) {

        super(mensagem);
    }
}

Listagem 22 – ContaNaoExisteException.java
```

5. Crie a Classe **OperacoesBanco.java** pelo windowBuilder como um JInternalFrame. Utilize o Design do window builder para criar conforme o protótipo. Deixe os atributos necessários com visibilidade para toda a classe, observe que alguns atributos estão estáticos, isto ocorre porque estes estão sendo acessados por métodos estáticos que os acessam ou modificam sem instanciar a classe. Siga a **listagem 23** abaixo.

```
package threeway.projeto.gui;

public class OperacoesBanco extends JInternalFrame {

    private static Cliente clienteSelecionado;
    private static ContaService contaService;
    private static Conta contaCliente;
    private static JPanel panelDadosConta;
    private static JTextField tfNumeroConta;
    private static JTextField tfDtAbertura;
    private static JTextField tfSaldo;
    private static JButton btnAbrirConta;
    private static JButton btnSaque;
    private static JButton btnDeposito;
    private static JButton btnTransferencia;
```

```
private static Boolean contaCriada = Boolean.FALSE;
private DialogAbrirConta dialogAbrirConta;
private DialogTransferencia dialogTransferencia;
private static JTable tableTransacoes;
private JPanel panelTransacoes;
}
```

Listagem 23 – OperacoesBanco.java

6. Inicialize as variáveis pelo método initialize(). Siga a **listagem 24** abaixo.

```
/**
 * Método responsável por inicializar componentes da tela
 */
private void initialize() {

    setClosable(true);
    setBounds(100, 100, 750, 375);

    setTitle("Operações Bancárias");
    getContentPane().setLayout(null);

    JLabel lblCliente = new JLabel("Cliente:");
    lblCliente.setFont(new Font("Tahoma", Font.BOLD, 12));
    lblCliente.setBounds(13, 25, 46, 23);
    getContentPane().add(lblCliente);

    JLabel lblNomeCliente = new JLabel(clienteSelecionado.getNome());
    lblNomeCliente.setFont(new Font("Tahoma", Font.BOLD, 12));
    lblNomeCliente.setBounds(69, 25, 155, 23);
    getContentPane().add(lblNomeCliente);

    panelDadosConta = new JPanel();
    panelDadosConta.setBorder(new TitledBorder(null, "Dados da
Conta", TitledBorder.LEADING, TitledBorder.TOP, null, null));
    panelDadosConta.setBounds(234, 11, 490, 53);
    getContentPane().add(panelDadosConta);
    panelDadosConta.setLayout(null);

    JLabel lblNumero = new JLabel("No :");
    lblNumero.setBounds(10, 20, 25, 14);
    panelDadosConta.add(lblNumero);

    tfNumeroConta = new JTextField();
    tfNumeroConta.setEditable(false);
    tfNumeroConta.setBounds(45, 17, 42, 20);
    panelDadosConta.add(tfNumeroConta);
    tfNumeroConta.setColumns(10);

    JLabel lblSaldo = new JLabel("Saldo:");
    lblSaldo.setBounds(361, 20, 36, 14);
    panelDadosConta.add(lblSaldo);

    JLabel lblDataDeAbertura = new JLabel("Data de Abertura:");
    lblDataDeAbertura.setBounds(110, 20, 101, 14);
    panelDadosConta.add(lblDataDeAbertura);

    tfDtAbertura = new JTextField();
    tfDtAbertura.setEditable(false);
    tfDtAbertura.setBounds(221, 17, 125, 20);
    panelDadosConta.add(tfDtAbertura);
    tfDtAbertura.setColumns(10);

    tfSaldo = new JTextField();
    tfSaldo.setEditable(false);
    tfSaldo.setBounds(394, 17, 86, 20);
    panelDadosConta.add(tfSaldo);
    tfSaldo.setColumns(10);

    btnAbrirConta = new JButton("Abrir Conta");
```

```

        btnAbrirConta.setBounds(293, 80, 101, 37);
        getContentPane().add(btnAbrirConta);

        btnSaque = new JButton("Saque");
        btnSaque.setBounds(504, 80, 90, 37);
        getContentPane().add(btnSaque);

        btnDeposito = new JButton("Deposito");
        btnDeposito.setBounds(404, 80, 90, 37);
        getContentPane().add(btnDeposito);

        btnTransferencia = new JButton("Transferência");
        btnTransferencia.setBounds(604, 81, 120, 36);
        getContentPane().add(btnTransferencia);

        JSeparator separator = new JSeparator();
        separator.setBounds(13, 70, 711, 10);
        getContentPane().add(separator);

        JSeparator separator_1 = new JSeparator();
        separator_1.setBounds(13, 128, 711, 10);
        getContentPane().add(separator_1);

        panelTransacoes = new JPanel();
        panelTransacoes.setBorder(new TitledBorder(null, "Historico de
Transa\u00E7\u00F5es",TitledBorder.LEADING, TitledBorder.TOP, null, null));
        panelTransacoes.setBounds(13, 149, 711, 186);
        getContentPane().add(panelTransacoes);
    }

    Listagem 24 – OperacoesBanco.java – initialize();

```

7. No construtor da classe **ManterCliente.java** chame o método **initialize()** e faça com que receba uma instancia de **ContaService.java** e **Cliente.java** como parâmetros em seu construtor. Siga a **listagem 25** abaixo.

```

/**
 * Responsável pela criação de novas instâncias desta classe.
 *
 * @param serviceConta
 *
 * @param cliente
 */
public OperacoesBanco( ContaService serviceConta, Cliente cliente ) {
    clienteSelecionado = cliente;
    contaService = serviceConta;
    initialize();
    verificaExistenciaDeContaParaCliente();
    iniciaJTable();
    atualizaDadosTabela();
    acaoBotaoAbrirConta();
    acaoBotaoDeposito();
    acaoBotaoSaque();
    acaoBotaoTransferencia();
}

    Listagem 25 – OperacoesBanco.java – Construtor

```

8. Crie os métodos que estão sendo chamados no construtor. Siga a **listagem 26** abaixo.

```

/**
 * Método responsável por verificar se já existe conta para o cliente

```

selecionado

```

*
*/
private void verificaExistenciaDeContaParaCliente() {
    Conta conta = contaService.buscaContaPorCliente(clienteSelecionado);
    if (conta != null) {
        mudaStatusContaCriada(Boolean.TRUE);
        contaCliente = conta;
    } else {
        mudaStatusContaCriada(Boolean.FALSE);
    }
    defineEnabledBotoes();
    preenchePanelDescricaoConta();
}

```

chamara o

```

/**
* Método responsável por adicionar evento ao botão de transferencia, este que

```

```

DialogTransferencia[JDialog]

```

```

*
*/
private void acaoBotaoTransferencia() {
    btnTransferencia.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            dialogTransferencia = new
            dialogTransferencia.setLocationRelativeTo(null);
            dialogTransferencia.setVisible(true);
        }
    });
}

```

DialogTransferencia(*contaService*, *contaCliente*);

```

/**
* Método responsável por adicionar evento ao botão de saque, este que chamará
para informar o valor do saque

```

um JOptionPane

```

*
*/
private void acaoBotaoSaque() {
    btnSaque.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                Double valorDeposito =
                Double.parseDouble(JOptionPane.showInputDialog(null, "Digite o valor para saque. (formato x00.00)",
                "Saque em conta bancária", JOptionPane.QUESTION_MESSAGE));
                contaService.sacar(contaCliente, valorDeposito);
                JOptionPane.showMessageDialog(null, "Saque
                realizado com sucesso!");
                preenchePanelDescricaoConta();
            } catch (SaldoInsuficienteException ex1) {
                JOptionPane.showMessageDialog(null,
                ex1.getMessage());
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(null, "Algum campo
                pode ter sido preenchido inválido. Verifique e tente novamente!");
            } finally {
                atualizaDadosTabela();
            }
        }
    });
}

```

```

/**
* Método responsável por adicionar evento ao botão deposito, este que chamará

```

um JOptionPane

```

para informar o valor do deposito
*
*/
private void acaoBotaoDeposito() {
    btnDeposito.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                Double valorDeposito =

```

```

Double.parseDouble(JOptionPane.showInputDialog(null, "Digite o valor para deposito. (formato x00.00)",
"Depósito em conta bancária", JOptionPane.QUESTION_MESSAGE));
        contaService.depositar(contaCliente,
                                valorDeposito);
        JOptionPane.showMessageDialog(null, "Deposito
        realizado com sucesso!");

        preenchePanelDescricaoConta();
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, "Algum campo
        pode ter sido preenchido inválido. Verifique e tente novamente!");
    } finally {
        atualizaDadosTabela();
    }
}

});

}

/**
 * Método responsável por adicionar evento ao botão de abrir conta, este que
chamara o
DialogAbrirConta[JDialog]
 */
private void acaoBotaoAbrirConta() {
    btnAbrirConta.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {
                dialogAbrirConta = new
                DialogAbrirConta(contaService, clienteSelecionado);
                dialogAbrirConta.setLocationRelativeTo(null);
                dialogAbrirConta.setVisible(true);
            } catch (ParseException e1) {
                e1.printStackTrace();
            }
        }
    });
}

/**
 * Método responsável por inicializar JTable
 */
private void iniciaJTable() {
    tableTransacoes = new JTable();
    tableTransacoes.setModel(new DefaultTableModel(new Object[][] {},
        retornaNomeColunas()));
    tableTransacoes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    tableTransacoes.setBounds(703, 24, -690, 140);
    tableTransacoes.getTableHeader().setReorderingAllowed(false);
    panelTransacoes.setLayout(null);
    JScrollPane scrollPane = new JScrollPane(tableTransacoes);
    scrollPane.setViewportView(tableTransacoes);
    scrollPane.setBounds(10, 21, 691, 154);
    panelTransacoes.add(scrollPane);
}

/**
 * Método responsável por retornar nome das colunas da JTable
 * @return String[]
 */
private static String[] retornaNomeColunas() {
    String[] columnNames = new String[5];
    columnNames[0] = ( "Tipo Transação" );
    columnNames[1] = ( "Titular/Conta Crédito" );
    columnNames[2] = ( "Titular/Conta Debito" );
    columnNames[3] = ( "Data" );
    columnNames[4] = ( "Valor" );
    return columnNames;
}

/**
 * Método responsável por Atualizar dados da tabela de transações sempre que
alguma transação e
feita esse metodo e chamado

```

```

*
*/
public static void atualizaDadosTabela() {
    tableTransacoes.setModel(new DefaultTableModel(new Object[][] {},
        retornaNomeColunas()));
    tableTransacoes.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    DefaultTableModel dtm = (DefaultTableModel) tableTransacoes.getModel();
    ArrayList<Transacao> listaTransacoes = new ArrayList<Transacao>();
    if (contaCliente != null) {
        listaTransacoes = new
ArrayList<Transacao>(contaCliente.getTransacoes());
    }
    Collections.sort(listaTransacoes);
    Iterator<Transacao> it = listaTransacoes.iterator();
    while (it.hasNext()) {
        Transacao tr = it.next();
        if (tr.getTipoTransacao() == EnumTipoTransacao.TRANSFERENCIA) {
            dtm.addRow(new Object[] { tr.getTipoTransacao(),
                tr.getContaCredito().getTitular().getNome() + "/" +
tr.getContaCredito().getNumero(),
                tr.getContaDebito().getTitular().getNome() + "/" +
tr.getContaDebito().getNumero(),
                tr.getValor() });
        } else {
            dtm.addRow(new Object[] { tr.getTipoTransacao(),
                tr.getContaCredito().getTitular().getNome() + "/" +
                "-", UtilData.DDMMAAAHHMM(tr.getData()),
                tr.getValor() });
        }
    }
}

/**
* Método responsável por definir se os botões abaixo definidos estarão
habilitados ou não
*
*/
public static void defineEnabledBotoes() {
    btnAbrirConta.setEnabled(!contaCriada);
    btnSaque.setEnabled(contaCriada);
    btnDeposito.setEnabled(contaCriada);
    btnTransferencia.setEnabled(contaCriada);
    panelDadosConta.setVisible(contaCriada);
}

/**
* Método responsável por mudar flag de contaCriada de acordo com a função
executada no
sistema
*
* @param flag
*/
public static void mudaStatusContaCriada(Boolean flag) {
    contaCriada = flag;
}

/**
* Método responsável por recuperar conta de um determinado cliente selecionado
*
*/
public static void recuperaContaCliente() {
    contaCliente = contaService.buscaContaPorCliente(clienteSelecionado);
    preenchePanelDescricaoConta();
}

/**
* Método responsável por preencher JTextField com os dados do cliente
selecionado
*
*/
public static void preenchePanelDescricaoConta() {
    if (contaCliente != null) {

```

```
tfDtAbertura.setText(UtilData.formataData(contaCliente.getDataAbertura()));
tfNumeroConta.setText(contaCliente.getNumero().toString());
tfSaldo.setText(contaCliente.getSaldo().toString());
    }
}
```

Listagem 26 – OperacoesBanco.java – Métodos

E. Figura 8 - Tela Abrir Conta

1. Crie a Classe **DialogAbrirConta.java** pelo windowBuilder como um JDialog. Utilize o Design do window builder para criar conforme o protótipo. Deixe os atributos necessários com visibilidade para toda a classe. Siga a **listagem 27** abaixo.

```
public class DialogAbrirConta extends JDialog {
    private ContaService service;
    private Cliente clienteSelecionado;
    private Conta conta;
    private JTextField tfSaldo;
    private JComboBox<EnumTipoConta> comboTipoConta;
    private JTextField tfNumero;
    private JButton btnCancelar;
    private JButton btnSalvar;
    private JLabel lblCamposObrigatorios;
    private JTextField txDtAbertura;
}
```

Listagem 27 – DialogAbrirConta.java

2. Inicialize as variáveis pelo método initialize(). Siga a **listagem 27** abaixo.

```
/**
 * Método responsável por inicializar componentes da tela
 *
 * @throws ParseException
 */
private void initialize() throws ParseException {
    setModal(true);
    this.setLocationRelativeTo(null);

    setTitle("Abertura de Conta");
    setBounds(100, 100, 300, 250);
    getContentPane().setLayout(null);

    JLabel lblNumero = new JLabel("Numero *");
    lblNumero.setBounds(10, 39, 80, 14);
    getContentPane().add(lblNumero);

    JLabel lblDataDeAbertura = new JLabel("Abertura:");
    lblDataDeAbertura.setBounds(10, 11, 80, 14);
    getContentPane().add(lblDataDeAbertura);

    JLabel lblSaldoInicial = new JLabel("Saldo Inicial:");
    lblSaldoInicial.setBounds(10, 67, 80, 14);
    getContentPane().add(lblSaldoInicial);

    btnCancelar = new JButton("Cancelar");
    btnCancelar.setBounds(175, 166, 99, 35);
    getContentPane().add(btnCancelar);

    btnSalvar = new JButton("Salvar");
    btnSalvar.setBounds(55, 166, 99, 35);
    getContentPane().add(btnSalvar);

    JSeparator separator = new JSeparator();
```



```

separator.setBounds(10, 153, 264, 2);
getContentPane().add(separator);

tfNumero = new JTextField();
tfNumero.setBounds(110, 36, 75, 20);
getContentPane().add(tfNumero);

tfSaldo = new JTextField();
tfSaldo.setBounds(110, 64, 75, 20);
getContentPane().add(tfSaldo);
tfSaldo.setColumns(10);

lblCamposObrigatrios = new JLabel("(*) Campos Obrigatórios");
lblCamposObrigatrios.setHorizontalAlignment(SwingConstants.RIGHT);
lblCamposObrigatrios.setFont(new Font("Tahoma", Font.PLAIN, 11));
lblCamposObrigatrios.setBounds(138, 120, 136, 24);
getContentPane().add(lblCamposObrigatrios);

txDtAbertura = new JTextField();
txDtAbertura.setBounds(110, 9, 170, 19);
txDtAbertura.setEnabled(false);
Date date = new Date();
txDtAbertura.setText(UtilData.formataData(date));
getContentPane().add(txDtAbertura);
txDtAbertura.setColumns(10);
}

```

Listagem 27 – DialogAbrirConta.java – initialize()

3. No construtor da classe **DialogAbrirConta.java** chame o método **initialize()** e faça com que receba uma instancia de **ContaService.java** e **Cliente.java** como parâmetros em seu construtor. Siga a **listagem 28** abaixo.

```

/**
 * Responsável pela criação de novas instâncias desta classe.
 *
 * @param service
 *
 * @param cliente
 *
 * @throws ParseException
 */
public DialogAbrirConta( ContaService service, Cliente cliente ) throws ParseException
{
    this.service = service;
    this.clienteSelecionado = cliente;
    this.conta = new Conta();
    initialize();
    tfSaldo.setText("0");

    caregaComboTiposConta();
    acaoBotaoCancelar();
    acaoBotaoSalvar();
}

```

Listagem 28 – DialogAbrirConta.java – Construtor

4. Crie os métodos que estão sendo chamados no construtor. Siga a **listagem 29** abaixo.

```

/**
 * Método responsável por adicionar evento ao botão salvar
 *
 */
private void acaoBotaoSalvar() {
    btnSalvar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            try {

```

```

montaObjetoConta();
service.salvar(conta);
JOptionPane.showMessageDialog(null, "Abertura de
conta realizada com sucesso!");

OperacoesBanco.mudaStatusContaCriada(Boolean.TRUE);

ex.getMessage();

ex.getMessage();

pode ter sido preenchido inválido. Verifique e tente novamente!");
}
}
});
}

/**
 * Método responsável por adicionar evento ao botão cancelar
 */
private void acaoBotaoCancelar() {
    btnCancelar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            setVisible(Boolean.FALSE);
        }
    });
}

/**
 * Método responsável por carregar combo com o EnumTiposConta
 */
private void caregaComboTiposConta() {
    for (EnumTipoConta tipoConta : EnumTipoConta.values()) {
        comboTipoConta.addItem(tipoConta);
    }
}

/**
 * Método responsável por montar objeto conta de acordo com os dados que o
usuario informar
 */
@author Wilker Machado
 */
@throws Exception
 */
public void montaObjetoConta() throws Exception {
    this.conta = new Conta(this.clienteSelecionado,
Integer.parseInt(tfNumero.getText()));
    this.conta.setSaldo(Double.parseDouble(tfSaldo.getText()));
    Date date = new Date();
    this.conta.setDataAbertura(date);
}
} Listagem 29 - DialogAbrirConta.java - Métodos

```

5. Veja que você precisará implementar a classe **NumeroDeContaJaExistenteException.java** para tratar a exceção quando o usuário informar um numero de conta que não existe. Siga a **listagem 30** abaixo.

package threeway.projeto.service.excecoes;

public class NumeroDeContaJaExistenteException **extends** Exception{

```

    public NumeroDeContaJaExistenteException() {
        super("Numero de conta ja existente.");
    }
}

```

```

    }

    public NumeroDeContaJaExistenteException(String mensagem) {
        super(mensagem);
    }
}
Listagem 30 - NumeroDeContaJaExistenteException.java - classe

```

F. Figura 9 - Tela Transferência entre Contas

1. Crie a Classe **DialogTransferencia.java** pelo windowBuilder como um JDialog. Utilize o Design do window builder para criar conforme o protótipo. Deixe os atributos necessários com visibilidade para toda a classe. Siga a **listagem 31** abaixo.

```

package threeway.projeto.gui;

public class DialogTransferencia extends JDialog {

    private TransacaoService transacaoService = new TransacaoService();
    private ContaService service;
    private Conta contaCliente;
    private Conta contaDestino;
    private JPanel panelDadosConta;
    private JTextField tfNumeroConta;
    private JTextField tfDtAbertura;
    private JTextField tfSaldo;
    private JTextField tfTitularConta;
    private JTextField tfNContaDestino;
    private JTextField tfValorTransferencia;
    private JButton btnCancelar;
    private JButton btnRealizarTransferencia;
    private JLabel lblCamposObrigatorios;
}
Listagem 31 - DialogTransferencia.java

```

2. Inicialize as variáveis pelo método initialize(). Siga a **listagem 32** abaixo.

```

/**
 * Método responsável por inicializar componentes da tela
 */
private void initialize() {

    setModal(true);
    this.setLocationRelativeTo(null);

    setTitle("Transferência entre contas");
    getContentPane().setLayout(null);
    setBounds(100, 100, 520, 270);

    panelDadosConta = new JPanel();
    panelDadosConta.setBorder(new TitledBorder(null, "Dados da Conta",
TitledBorder.LEADING, TitledBorder.TOP, null, null));
    panelDadosConta.setBounds(10, 11, 484, 77);
    getContentPane().add(panelDadosConta);
    panelDadosConta.setLayout(null);

    JLabel lblNumero = new JLabel("No :");
    lblNumero.setBounds(10, 20, 25, 14);
    panelDadosConta.add(lblNumero);

    tfNumeroConta = new JTextField();
    tfNumeroConta.setEditable(false);
    tfNumeroConta.setBounds(34, 17, 42, 20);
    panelDadosConta.add(tfNumeroConta);
}

```

```
tfNumeroConta.setColumns(10);

JLabel lblSaldo = new JLabel("Saldo:");
lblSaldo.setBounds(343, 20, 36, 14);
panelDadosConta.add(lblSaldo);

JLabel lblDataDeAbertura = new JLabel("Data de Abertura:");
lblDataDeAbertura.setBounds(97, 20, 101, 14);
panelDadosConta.add(lblDataDeAbertura);

tfDtAbertura = new JTextField();
tfDtAbertura.setEditable(false);
tfDtAbertura.setBounds(208, 17, 125, 20);
panelDadosConta.add(tfDtAbertura);
tfDtAbertura.setColumns(10);

tfSaldo = new JTextField();
tfSaldo.setEditable(false);
tfSaldo.setBounds(388, 17, 86, 20);
panelDadosConta.add(tfSaldo);
tfSaldo.setColumns(10);

JLabel lblTitularDaConta = new JLabel("Titular da Conta:");
lblTitularDaConta.setBounds(10, 48, 101, 14);
panelDadosConta.add(lblTitularDaConta);

tfTitularConta = new JTextField();
tfTitularConta.setEditable(false);
tfTitularConta.setBounds(121, 45, 353, 20);
panelDadosConta.add(tfTitularConta);
tfTitularConta.setColumns(10);

JPanel panel = new JPanel();
panel.setBorder(new TitledBorder(null, "Dados para Transfer\u00EAncia",
    TitledBorder.LEADING, TitledBorder.TOP, null, null));
panel.setBounds(10, 99, 484, 77);
getContentPane().add(panel);
panel.setLayout(null);

JLabel lblNContaDestino = new JLabel("N\u00BA Conta Destino *");
lblNContaDestino.setBounds(10, 24, 140, 14);
panel.add(lblNContaDestino);

JLabel lblValorDaTransferencia = new JLabel("Valor da Transfer\u00EAncia *");
lblValorDaTransferencia.setBounds(10, 49, 149, 14);
panel.add(lblValorDaTransferencia);

tfNContaDestino = new JTextField();
tfNContaDestino.setBounds(160, 21, 86, 20);
panel.add(tfNContaDestino);
tfNContaDestino.setColumns(10);

tfValorTransferencia = new JTextField();
tfValorTransferencia.setBounds(160, 46, 86, 20);
panel.add(tfValorTransferencia);
tfValorTransferencia.setColumns(10);

lblCamposObrigatorios = new JLabel("(*) Campos Obrigat\u00F3rios");
lblCamposObrigatorios.setHorizontalAlignment(SwingConstants.RIGHT);
lblCamposObrigatorios.setBounds(315, 49, 159, 14);
panel.add(lblCamposObrigatorios);

btnCancelar = new JButton("Cancelar");
btnCancelar.setBounds(343, 187, 151, 33);
getContentPane().add(btnCancelar);

btnRealizarTransferencia = new JButton("Realizar Transfer\u00EAncia");
btnRealizarTransferencia.setBounds(172, 187, 161, 33);
getContentPane().add(btnRealizarTransferencia);
}
```

Listagem 32 – DialogTransferencia.java – initialize()

- 3.** No construtor da classe **ManterCliente.java** chame o método **initialize()** e faça com que receba uma instancia de **ContaService.java** e **Conta.java** como parâmetros em seu construtor. Siga a **listagem 33** abaixo.

```
/**
 * Responsável pela criação de novas instâncias desta classe.
 *
 * @param contaService
 *
 * @param contaCliente
 */
public DialogTransferencia( ContaService contaService, Conta contaCliente ) {
    this.service = contaService;
    this.contaCliente = contaCliente;
    initialize();
    preenchePanelDescricaoConta();
    acaoBotaoCancelar();
    acaoBotaoRealizarTransferencia();
}
```

Listagem 33- DialogTransferencia.java – Construtor

- 4.** Crie os métodos que estão sendo chamados no construtor. Siga a **listagem 34** abaixo.

```
/**
 * Método responsável por adicionar evento ao botão de realizar transferencia
 *
 */
private void acaoBotaoRealizarTransferencia() {

    btnRealizarTransfercia.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            try {

                int numeroConta = Integer.parseInt(tfNContaDestino.getText());

                Conta contaDestino = service.buscaContaPorNumero(numeroCon-
ta);

                double valorTransferencia = Double.parseDouble(tfValorTranfer-
encia.getText());

                service.transferir(contaCliente, valorTransferencia, contaDestino);

                OperacoesBanco.atualizaDadosTabela();

                OperacoesBanco.preenchePanelDescricaoConta();

                JOptionPane.showMessageDialog(null, "Transferência realizada
com sucesso!");

                setVisible(Boolean.FALSE);

            } catch (ContaNaoExisteException exConta) {

                JOptionPane.showMessageDialog(null, exConta.getMessage());

            } catch (SaldoInsuficienteException ex1) {

                JOptionPane.showMessageDialog(null, ex1.getMessage());

            }

        }

    });
}
```

```
        } catch (Exception ex) {  
            JOptionPane.showMessageDialog(null, "Algum campo pode não ter  
            sido preenchido ou preenchido inválido. Verifique e tente novamente!");  
        }  
    }  
});  
}  
  
/**  
 * Método responsável por adicionar evento ao botão de cancelar  
 */  
private void acaoBotaoCancelar() {  
    btnCancelar.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent e) {  
            setVisible(Boolean.FALSE);  
        }  
    });  
}  
  
/**  
 * Método responsável por preencher JPanel com as informações da contaCliente  
 */  
private void preenchePanelDescricaoConta() {  
    if (contaCliente != null) {  
        tfDtAbertura.setText(UtilData.formataData(contaCliente.getDataAbertura()));  
        tfNumeroConta.setText(contaCliente.getNumero().toString());  
        tfSaldo.setText(contaCliente.getSaldo().toString());  
        tfTitularConta.setText(contaCliente.getTitular().getNome());  
    }  
}
```

Listagem 34- DialogTransferencia.java – Métodos